THINK Project Manager Menus 30

This chapter describes each of the THINK Project Manager menu commands. It is organized by menu, from left to right along the menu bar. Within each menu, commands are described in the order in which they appear in the menu.

Contents

The Apple Menu 611
About THINK Project Manager 611

The File Menu 611
New/
New Project 611
Open/
Open Project 613
Open Selection 614
Close 614
Save 614
Save As 614
Save a Copy As 615
Revert 615
Page Setup 615
Print 615
Modify Read-Only 615
Quit 615

The Edit Menu 616
Undo 616
Cut 616
Copy 617
Paste 617
Clear 617
Select All 617
Options 617
THINK Project Manager options 619
Preferences page 620
Editor page 622
Debugging page 624
Extensions page 626

Project Window page 627
.o Converter options 628
THINK C options 628
Language Settings Page 629
Compiler Settings page 633
Code Optimization page 639
Debugging page 642
Prefix page 644
THINK Rez options 645
Set tabs & font 646
Shift Left 647
Shift Right 647
Balance 647

The Search Menu 648
Find 648
Enter Selection 650
Find Again 650
Replace 650
Replace & Find Again 651
Replace All 651
Find in Next File 651
Find in THINK Reference 652
Go To Line 652
Mark 653
Remove Marker 654
Go To Next Error 654
Go To Previous Error 654

The Project Menu 655
Close Project 655
Close & Compact 655
Switch to Project 655
Set Project Type 655
Application dialog 656
Desk Accessory and Device Driver dialogs 658
Code Resource dialog 659
Remove Objects 660
Bring Up To Date 660
Check Link 660
Build Library 661
Build Application
Build Desk Accessory
Build Device Driver
Build Code Resource 661
Use Debugger 662
Run 662

The Source Menu 663
Add Window 663
Add Files 664
Remove 664
Get Info 665
Debug 665
SourceServer 666
Check Syntax 667
Preprocess 667
Disassemble 667
Precompile 667
Compile
Load Library 667
Make 668
Browser 670

The Windows Menu 671
Arrange 671
Zoom 672
Full Titles 672
Close All 672
Save All 672
Project window 672
Edit Windows 672

The Apple Menu

About THINK Project Manager

This command tells you what version of THINK Project Manager you're using.

The File Menu

*Use the **File** menu commands to work with files that you open and edit with the THINK text editor. This menu also contains the commands that let you print open windows as well as quit the THINK Project Manager.*

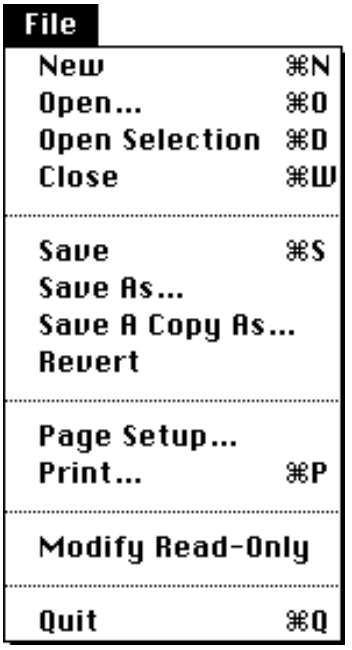

Figure 30-1 File menu

New/ New Project

If a project is open, this command appears as **New**. With no open project, this command is **New Project**.

The **New** command opens a new untitled window. You must save this file with one of the extensions defined on the Extensions page of the **THINK Project Manager options** dialog if you plan to compile it or add it to the project.

The **New Project** command creates a new project document. It displays a dialog like the one in Figure 30-2. If you have THINK C, this dialog contains different choices. The dialog contains a list of the available project models. If the Create folder option is checked, the THINK Project Manager creates a new folder in which to put the

new project document and other files. Otherwise, it puts the files in the folder you select in the next dialog.

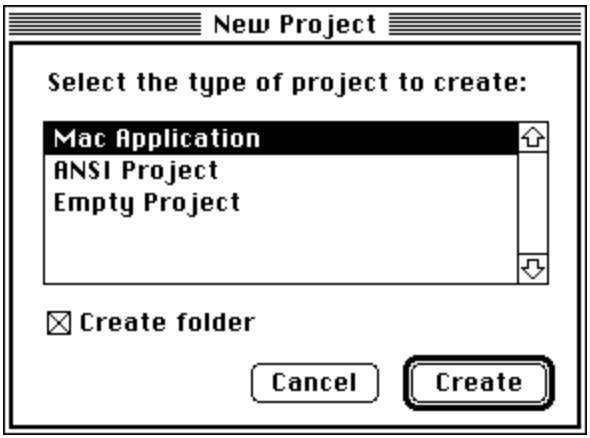

Figure 30-2 New Project dialog

You can choose from the project models in Table 30-1. Symantec C++ includes more models, which are described in the *Symantec C++ Compiler Guide* and the *Visual Architect and THINK Class Library Guide.*

This model…	Is for…
Mac Application	A program that uses the Macintosh Toolbox. The project model includes an empty source file named `main.c`. The project document includes the MacTraps library and `main.c`
ANSI Project	A program that uses the ANSI library or reads from and writes to the console window. The project model includes a source file named `main.c` that contains a simple `main()` routine and includes `stdio.h`. The project document includes the ANSI library and `main.c`.
Empty Project	A program that doesn't fall into the other two categories. The project model contains no source files. The project document is empty.

Table 30-1 THINK C project models

To choose a project model, double-click it, or select it and click Create.

Next, you see a standard file dialog, as in Figure 30-3. Enter a name for the project and click Save.

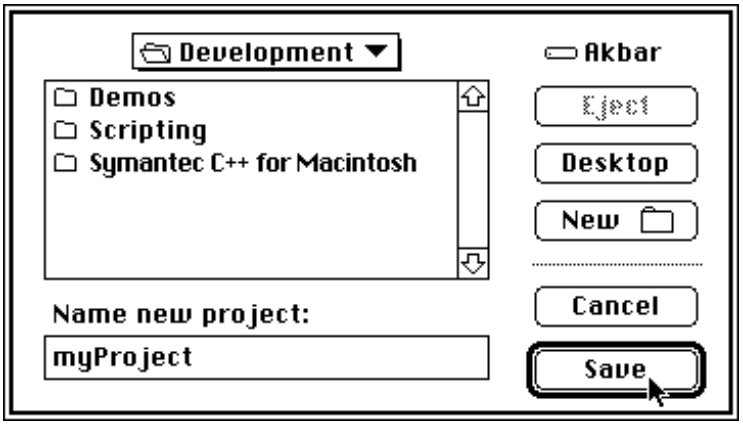

Figure 30-3 Naming the project

The THINK Project Manager creates a project document with your project name and the suffix `.π`. If the project model contains any source files, the THINK Project Manager creates copies of those. If you checked the Create folder option, it puts these files in a folder with your project name and the suffix `ƒ`. The THINK Project Manager opens the project document and displays the Project window.

Open/ Open Project

If a project is open in the THINK Project Manager, this command appears as **Open**. With no project open, this command appears as **Open Project**.

Open Project displays a dialog box that lets you select from the existing project documents on the disk and open one.

The **Open** command displays a dialog box that lets you select from existing text files on the disk and open one for editing. When you first begin a project, one way to add a file is to open it with this command, then compile it or add it with the **Add Files** command from the **Source** menu. The **Add Files** command lets you add multiple files without compiling or opening the files. Once files have been added to the project, they can be opened by double-clicking the file name in the Project window.

You can open multiple files. The edit windows stacks up with the titles showing, so you can easily click on any window to bring it to the front.

If you open too many files at once, you may have to close some windows to free up memory when you compile.

Open Selection

This command lets you open an included header file simply by selecting its name in the current program text. You don't need to select the `.h` extension (or the full path name). The selection is automatically extended to the right to include it as long as the file name itself is selected.

Close

This command has the same effect as clicking on the active window's close box. If you try to close an edit window and the file has been modified since it was last saved, a dialog box asks you if you want to save the changes, discard them, or cancel the **Close** command. To close the project window, use the **Close Project** command from the **Project** menu.

If the Confirm Saves option (in the **THINK Project Manager options** dialog) is off, the THINK Project Manager saves all changed files (except for untitled files) without asking.

Save

This command saves the file in the active edit window to disk. If the file is currently untitled, a dialog box asks you to name the file.

Note that an updated file can be compiled and added to the project without being saved, as long as it has been saved at least once and given a name with one of the extensions defined on the Extensions page of the **THINK Project Manager options** dialog.

Save As

This command lets you save the current file under another name. If you have made edits in the current session, they are saved under the new name. The original file remains unchanged and, as you continue editing, you will be editing the new file. This feature is useful if you want to work with a new version of a file, while keeping the old file as a backup.

Save As tries to preserve the tie between the file you are editing and its entry in the Project window. The entry for the file in the Project window is changed to match the new file name, if the file appears in the Project window, the name you give it has one of the extensions on the Extensions page of the **THINK Project Manager options**

dialog, and the new name doesn't already appear in the project window. Use **Save a Copy As** if you don't want this to happen.

Save a Copy As

Unlike **Save As**, this command does not affect the status of the file currently being edited; it simply snapshots it to another file. This is a good way to make backups without finding yourself editing the backup.

Neither **Save As** nor **Save a Copy As** lets you save into a file already open in an edit window.

Revert

This command restores the last-saved version of the current file and discards any edits made since the last save.

Page Setup

This command displays the standard **Page Setup** dialog that lets you specify the size of the paper you're printing on, and whether the file should be printed upright on the page (tall orientation) or sideways (wide orientation). See your Macintosh owner's manual for details.

Print

This command lets you print the current file, Project window, Search Results window, Compile Errors window, or Link Errors window. The **Print** dialog box lets you set the page range as well as other options, such as the Print Pages in Reverse Order option. When you click OK, your file prints. Each page of the file has a header showing the name of the file and the last modification date. The thumb of the vertical scroll bar moves from top to bottom (when printing in forward order) and from bottom to top (when printing in reverse order), to show you the printing progress. To cancel printing, press Command-Period.

Modify Read-Only

This command lets you modify a file that SourceServer or MPW Projector has marked read-only. This command is available only if you turn on the Projector-Aware option (on the Editor page of the **THINK Project Manager options** dialog).

Quit

This command exits the THINK Project Manager and returns to the Finder.

The Edit Menu

*The **Edit** menu contains the standard Macintosh editing commands (**Cut**, **Copy**, **Paste**) as well as other commands for formatting your source files. The **Edit** menu also contains the **Options** hierarchical menu, which lets you set options for the THINK Project Manager and each of its translators.*

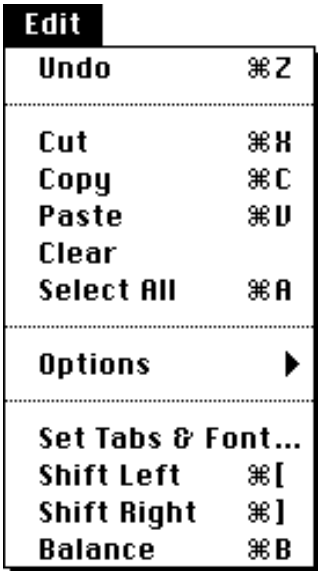

Figure 30-4 Edit menu

Undo

The **Undo** command reverses the last edit operation. The name of this command changes to reflect the operation you are undoing. After a **Paste**, for instance, this command changes to **Undo Paste**. Once you've undone something, the name of this command changes to **Redo**.

If nothing exists to undo, this command is dimmed. If the operation to undo doesn't belong to the frontmost window, the name of the command reflects the appropriate editing operation that can be undone, but the command is dimmed.

You can't undo a **Replace All**, **Revert**, or a **Set Tabs & Font** command.

Cut

This command removes selected text and places it in the Clipboard. It replaces the current contents of the clipboard (if there are any). Use the **Paste** command to insert text from the Clipboard into your file at the insertion point.

Copy

This command copies the selected text and places it in the Clipboard. The copy can now be pasted somewhere else using the **Paste** command.

Paste

This command copies the contents of the Clipboard into the file being edited at the insertion point. If text is currently selected, it is replaced by the Clipboard contents.

Clear

This command clears the selected text, but the cleared text is not placed on the Clipboard. The Clear key on your keyboard has the same effect as the **Clear** command.

Select All

This command selects all the text in the current edit window.

Options

This command gives you a lot of control over how the THINK Project Manager behaves and how its translators compile your code. When you choose this command, you see a hierarchical menu with four items (five if you have Symantec C++). The first item is for the THINK Project Manager and lets you determine how it behaves. The other items are for translators and let you determine how they compile code.

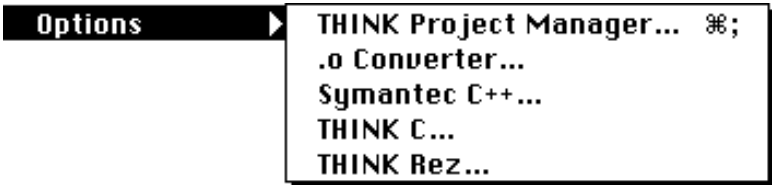

Figure 30-5 Options hierarchical menu

Note

If you have Symantec C++, see the *Symantec C++ Compiler Guide* for information on the **Symantec C++ options** dialog.

When you select one of the items in the hierarchical menu, you'll see a dialog like the one in Figure 30-6.

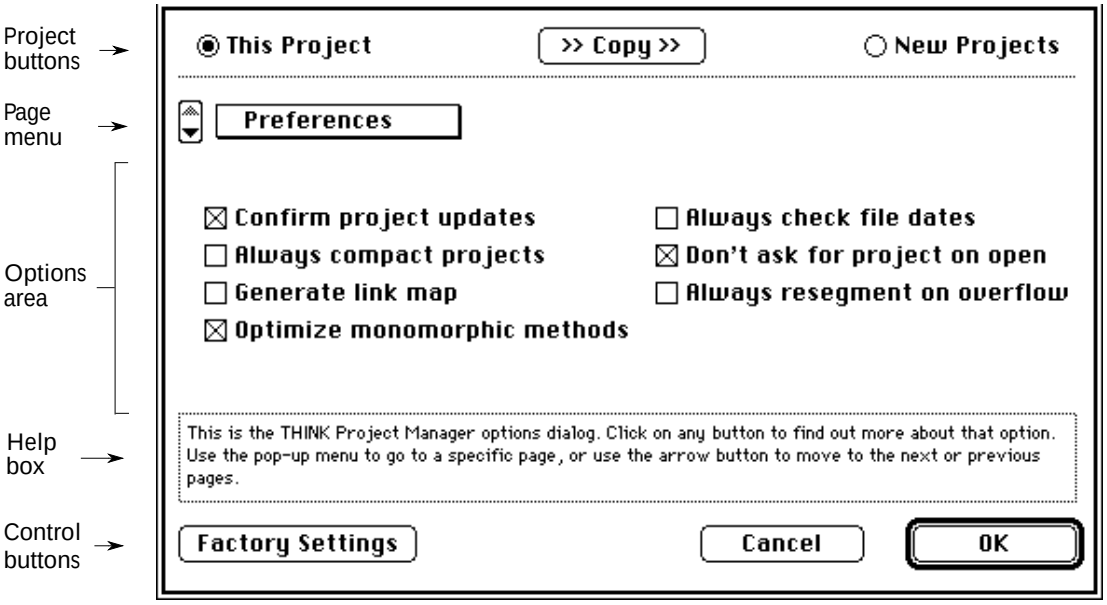

Figure 30-6 THINK Project Manager options dialog

You set your options in the options area in the middle. The options shown in this area vary, depending on the page of the dialog displayed. The other sections appear the same in all the options dialogs.

The help box near the bottom displays information about an option when you click that option. To get help on an option without changing its setting, click it without releasing the mouse button, move the mouse off the option, then release the button.

When an options dialog contains a lot of options, they are displayed on separate pages. The page menu in the upper-left corner lets you choose among them. Click the pop-up menu and select the page name. You can also scroll through the different pages with the arrow keys or the arrow buttons to the left of the pop-up menu.

The project buttons at the top let you choose whether you're setting options for the current project or defaults for use on all new

projects. Use the Copy button at the top of the dialog box to copy the defaults to the current project, or if you want the options you've set for a particular project to be the THINK Project Manager options. Note that even though only one page of the options shows up on the screen at one time, the Copy button copies *all* of the options, even the ones on the pages you can't see. However, it doesn't affect settings in other option dialogs.

The control buttons are at the bottom. When you click the Factory Settings button, the THINK Project Manager sets all the options on the pages back to their factory settings, which are the settings they had when you first installed the THINK Project Manager.

When you click OK, the changes for all pages of the dialog are saved. When you click Cancel, the changes for all the dialog pages are discarded.

THINK Project Manager options

The **THINK Project Manager options** dialog contains these five pages:

- Preferences, with options for specifying how THINK Project Manager behaves when it rebuilds projects and opens and closes files.
- Editor, with options for choosing how the THINK Project Manager handles search and replace operations, whether to use an external file editor, and details concerning saving and reopening files.
- Debugging, for specifying how the THINK Debugger works.
- Extensions, for mapping filename extensions to translators.
- Project Window, where you can choose to receive size information for each file and specify whether to show the segment number for each segment.

Preferences page

*The Preferences page of the **THINK Project Manager options** dialog lets you set how the THINK Project Manager builds projects, opens files, and closes files.*

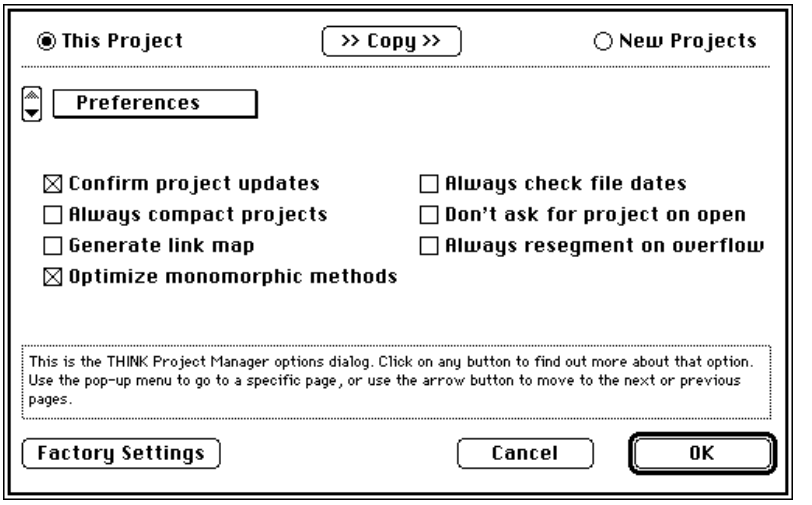

Figure 30-7 Preferences page

Confirm project updates
With this option off, the THINK Project Manager does not display the Bring project up to date? dialog when you choose the **Run**, **Check Link**, or any of the **Build** commands. The THINK Project Manager assumes you would have selected Update. The factory setting is on.

Always compact projects
If you turn this option on, the project document is compacted when you close the project or quit (but not when you run your application from the THINK Project Manager). The factory setting is off.

To achieve its remarkable speed, the THINK Project Manager pre-allocates space in the project for anticipated requirements. It does not necessarily free up the space when an item is deleted. As much as 20% of an uncompacted project document can contain unused space.

Compaction may be time-consuming, so you normally want this option on only when disk space is at a premium. The amount of space freed will vary. To compact a project without setting this option, use the **Close & Compact Project** command from the **Project** menu.

Generate link map
With this option on, the THINK Project Manager writes a link map for your application when you build it. The factory setting is off.

The link map lists all your project's segments, including one for global data. For each function (or global variable) in a segment, the map lists its name, its position in the segment, and the file in which it is defined.

The THINK Project Manager creates the link map only when you use the **Build Application** command. The name of the map is the name of the project with `.map` appended. For example, the link map for the `Bullseye.π` project is `Bullseye.π.map`. THINK Project Manager places the link map in the project folder and erases any link map that was there.

A monomorphic method is one that does not override another method and is never overridden itself.

Optimize monomorphic methods
If this option is on, THINK Project Manager optimizes monomorphic methods by generating a call directly to the method. If this option is off, THINK Project Manager generates a call to a run-time dispatcher.

You rarely need to turn this option off. Keep it on unless you're working with a program that has replaceable code modules.

Always check file dates
If this option is on, THINK Project Manager checks the modification time for each file in your project whenever you try to run or build it. Outdated files will be flagged for compilation. This is like choosing the Use Disk option in the **Make** dialog.

Don't ask for project on open
If this option is off, the THINK Project Manager asks you to open a project when you first start it or when you close a project. If this option is off, the THINK Project Manager doesn't prompt you. The factory setting is off.

Always resegment on overflow
This option controls what happens when the THINK Project Manager is compiling a project and the project contains a segment over 32K. With this option off, the THINK Project Manager displays a dialog asking if you want it to resegment the project. With this option on, the THINK Project Manager resegments the project without first asking you.

Editor page

The Editor page lets you specify how THINK Project Manager handles search and replace options, and whether or not to confirm saves, reopen files, and use an external editor.

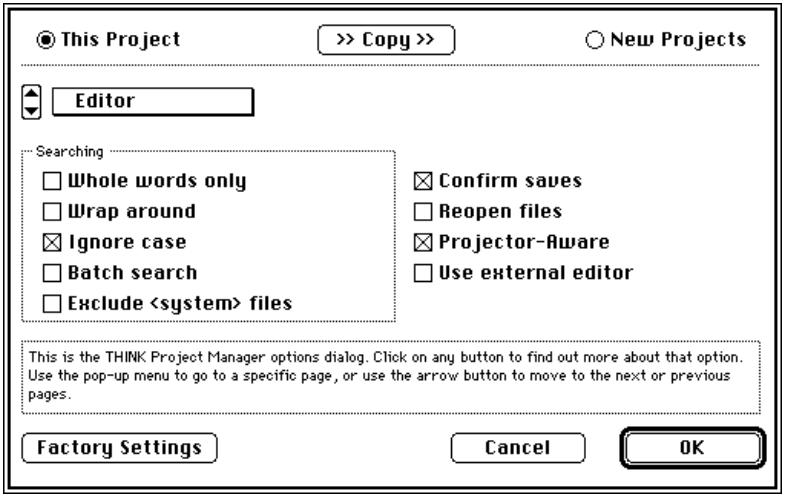

Figure 30-8 Editor page

Whole words only
With this option on, the editor's search and replace functions work on whole words only. For example, a search for "string" does not match the word "strings." The factory setting is off.

When you set this option in the **Find** dialog, it applies only to the current session. The **Find** command is described later in this chapter.

Wrap around
With this option on, the editor's search and replace functions search the entire file rather than only from the current position to the end of the file. When the end of the file is reached, the THINK Project Manager "wraps around" to the beginning of the file, flashes the menu bar, and continues. The factory setting is off.

Ignore case
With this option on, the editor's search and replace functions disregard case when performing a search. A search string matches either upper- or lowercase. The factory setting is on.

Batch search
With this option on, the editor's multi-file search function finds all matching occurrences of the search pattern and displays them in a window, instead of stopping at the first matching occurrence.

Exclude <system> files
With this option on, the editor's multi-file search function will not search through files that are in the THINK Project Manager tree.

Confirm saves
With this option off, THINK Project Manager automatically saves changes to a file that you have modified without prompting you for confirmation. The factory setting is on.

Turning this option off is dangerous, since it protects you from inadvertently replacing previous versions of your files with newly modified versions. However, it is very convenient when you want to do program development in quick, incremental steps.

Reopen files
With this option on, the THINK Project Manager remembers which project files were open when you close the project, and it reopens them the next time the project is opened. The THINK Project Manager does not remember files outside the project in this way.

Projector-Aware
When this option is on, THINK Project Manager adds the **Modify Read-only** command to the **File** menu and checks for a SourceServer (or Projector) resource (`'ckid'`) every time it opens a source file. THINK Project Manager displays an icon in the lower-left corner of the edit window that tells you whether SourceServer (or Projector) marked the file as read-only. If the file is a read-only file, its window displays a crossed-out pencil icon. If it's not a read-only file, its window displays a regular icon. If it's a modified read-only file, its window displays a pencil icon with a dotted cross-out.

Use external editor
If you turn on this option and you are running under System 7, THINK Project Manager uses the editor tool to open files for editing.

Debugging page

The Debugging page lets you specify how the THINK Debugger works.

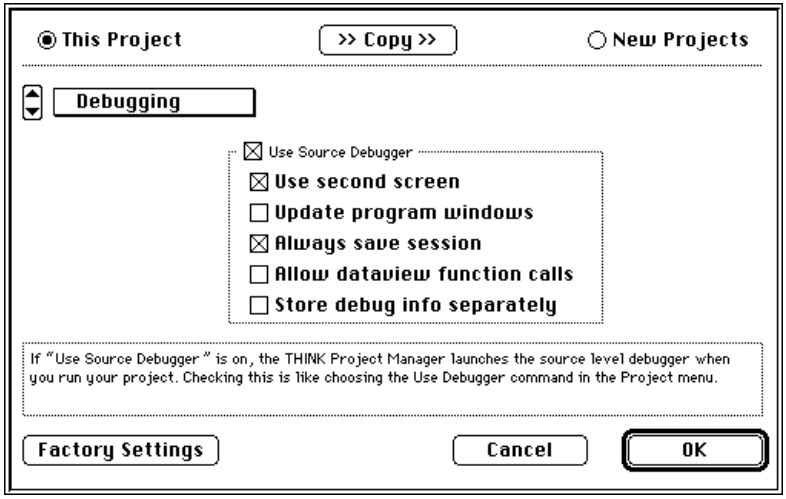

Figure 30-9 Debugging page

Use source debugger
When this option is checked, THINK Project Manager launches the source-level debugger when you run your project. This option is a "master switch." If it's off, the options below it are disabled. If it's on, the options below it are enabled, and set to the values they had the last time the master switch was on. The next three sections describe those options.

Checking this option has the same effect as choosing the **Use Debugger** command from the **Project** menu.

Use second screen
When this option is on and you're running on a Macintosh with more than one screen, THINK Project Manager displays the source debugger windows on the second screen.

Update program windows
With this option on, the debugger tries to update your windows when your project is stopped. The factory setting is off.

With this option off, your program must wait until control comes back so it can handle updates itself. Because it cannot do so until it gets back to its event loop, an update may remain pending for some time.

This option is especially useful when you're trying to step through code that draws in a window. This option uses memory to save your window's image. If there isn't enough memory, the debugger won't perform automatic updates.

Always save session
When this option is on, the debugger always saves your breakpoints and the contents of your data windows when you exit the debugger. When this option is off, you must choose **Save** from the debugger's **File** menu to save your session.

Allow dataview function calls
When this option is on, the debugger lets you put expressions that contain function calls into the Data window.

Warning
Use this option carefully. Use only functions that have no side effects and that will always return.

Store debug info separately
With this option on, the debug information for each source file is stored in a separate symbol file instead of in the project. This option can be used to keep very large projects under the 16MB limit.

Extensions page

The Extensions page lets you set the file extensions that appear at the end of the names of your source files.

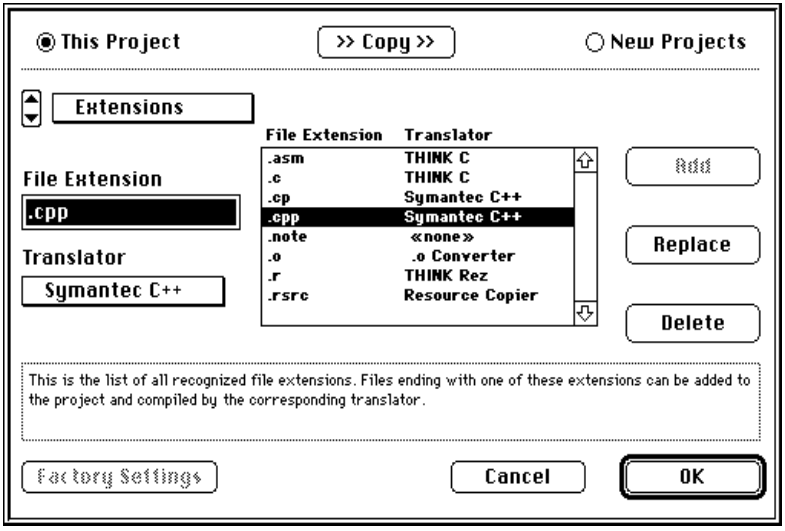

Figure 30-10 Extensions page

Add
Clicking this button adds a new file extension to the File Extension list and associates it with the selected translator.

Replace
Clicking this button replaces the selected extension in the File Extension list with a file extension you enter. The new extension is associated with the translator you choose.

Delete
Clicking this button deletes a selected file extension from the File Extension list. If you delete a file extension, you can no longer use files with that extension in the project.

Project Window page

*The **Project Window** options give you size information for each file and specify whether to show the segment number for each segment.*

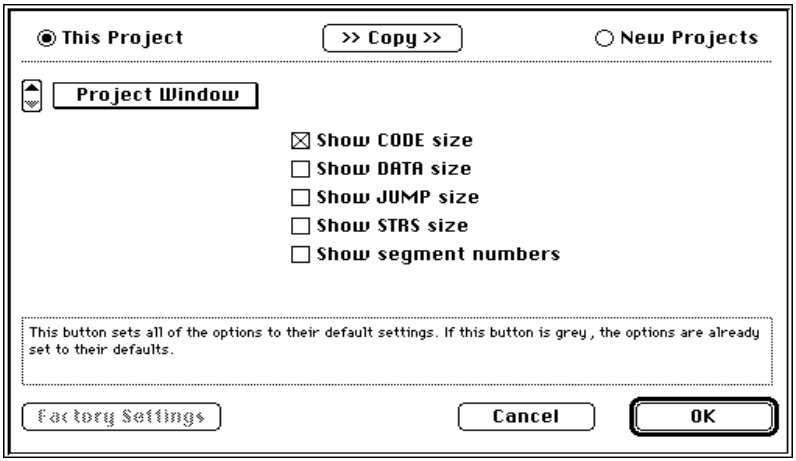

Figure 30-11 Project Window page

Show CODE size
With this option on, the Project Window shows the code size of each file in the project in a column to the right of the filenames.

Show DATA size
Turning this option on allows you to see each file's contribution to the data segment.

Show JUMP size
Turn this option on to display how much JUMP table space each file uses.

Show STRS size
With this option on, the Project Window shows each file's contribution to the string segment. If the Separate STRS option is off, this value is always 0 (zero).

Show segment numbers
Turn this option on to have the Project Window display the segment number of each segment.

.o Converter options

The **.o Converter options** dialog contains choices for setting vocabulary options.

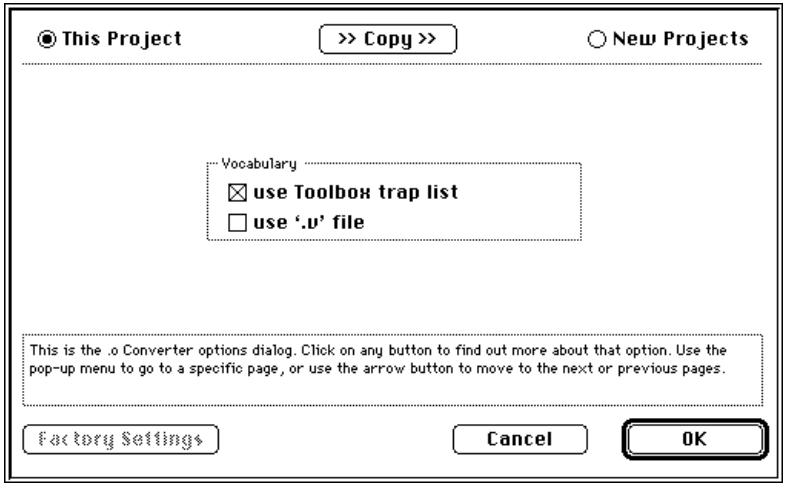

Figure 30-12 .o Converter options dialog

use Toolbox trap list
If this option is on, THINK Project Manager converts any uppercase name that is the same as a Toolbox Trap name to the *Inside Macintosh* spelling of the Toolbox Trap.

use '.v' file
If this option is on, THINK Project Manager converts any uppercase name that is the same as a name in the supplied vocabulary file to the vocabulary file's mixed-case version of the spelling. If the vocabulary file does not exist, the THINK Project Manager creates one that you can edit to provide the desired capitalization.

THINK C options

The **THINK C options** dialog has these five pages of options:

- Language Settings, for choosing which extensions to the C language you can use.
- Compiler Settings, for controlling how THINK C generates code.
- Code Optimization, for controlling how THINK C optimizes your code.

- Debugging, for stack frame and profiler call generation, as well as the choice of long- or short-format MacsBug names.
- Prefix, for writing preprocessor code that THINK C will include in all your files.

Language Settings Page

The Language Settings page lets you choose whether the THINK C compiler uses some extensions to the C language.

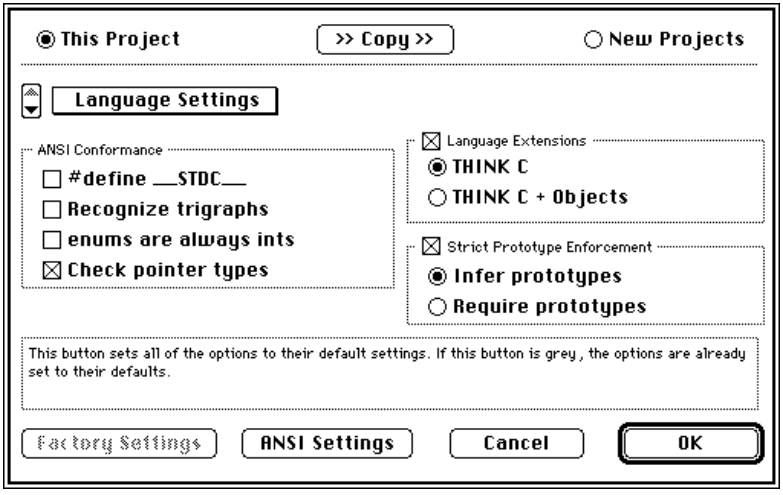

Figure 30-13 Language Settings page

All the options on this page control whether THINK C is ANSI-conformant. This is the only page that contains the ANSI Settings button. When you click that button, the options on this page are set so that THINK C is ANSI-conformant. The button is grayed out if THINK C is already ANSI-conformant.

#define __STDC__
If this option is on, THINK C defines the macro `__STDC__`. If it's off, THINK C doesn't define `__STDC__` at all. The factory setting is off. The ANSI-conformant setting is on.

If THINK C is ANSI-conformant, `__STDC__` is defined to be `1`. If THINK C isn't ANSI-conformant, `__STDC__` is defined to be `0`.

Recognize trigraphs
If this option is on, you can use trigraph sequences in your code. The factory setting is off. The ANSI-conformant setting is on.

Turn this option on only if you're writing strictly conformant ANSI C programs. Many Macintosh programs use character strings that resemble trigraphs. For example, this file type:

```
'????'
```

is interpreted as:

```
'?^
```

THINK C replaces the trigraph `??'` with `^` and complains that the character literal lacks a closing apostrophe (`'`). To write that file type with the Recognize trigraphs option on, use this:

```
'???\?'
```

enums are always ints
When this option is on, all enumeration constants are the same size as an `int`. You cannot use constants as large as a `long int`. When this option is off, enumeration constants can be the same size as a `char`, `short int`, `int`, or `long int`. The factory setting is off. The ANSI-conformant setting is on.

If this option is off, THINK C makes enumeration constants as small as possible. And, if necessary, it will make a constant as large as a `long int`. For example, these constants will only be as large as a `char`:

```
enum { red=1, yellow, green };
```

And these constants will be as large as a `long int`:

```
enum {
   million=1000000, billion=1000000000
};
```

Unless you're writing ANSI-conformant code, leave this option off.

Check pointer types
With this option on, THINK C makes sure that pointer types match when you assign one pointer to another or when you do pointer arithmetic. With this option off, THINK C treats all pointers as equivalent types and won't display the `pointer types do not match` error message. When you are subtracting two pointers, however, the two types must be pointers to objects of the same size.

Language extensions
This option lets you use THINK C's extensions to the C language. You can choose between two sets of extensions: THINK C or THINK C + Objects. If you don't select this option, you can use only ANSI C with no extensions. To be ANSI-conformant, turn this option off.

The THINK C extensions to the C language make programming on the Macintosh easier. They include inline assembly, the `pascal` keyword, and `//` comments. The object extensions let you write object-oriented code in C. See Chapter 23, "Using Objects in THINK C."

Note

THINK C + Objects is *not* C++. It is a subset of C++ that includes most of what you need for object-oriented programming, but does not include operator overloading, multiple inheritance, and some other features. For a full implementation of C++, use Symantec C++.

If you use the object extensions, these words become keywords, and you can't use them as identifiers:

```
class                delete
new                  virtual
```

To relax prototype checking when you precompiled a header file, turn off the Check pointer types option.

Strict prototype enforcement
This option lets you choose how strictly THINK Project Manager enforces the use of prototypes. If this option is on, you can choose between two enforcement levels: Infer prototypes and Require prototypes. If this option is off, THINK Project Manager does nothing when you use a function without a prototype. The factory setting is Infer prototypes, which is also the ANSI-conformant setting.

This table explains the two enforcement levels:

If you choose…	When you use a function without a prototype, THINK Project Manager…
Infer prototypes	Infers a prototype from the first appearance of a function. That appearance can be a function call or an old-style declaration. If a subsequent call, declaration, or prototype doesn't match the inferred prototype, THINK Project Manager raises an error.
Require prototypes	Raises an error. You can't use or define a function unless it has a prototype. New-style function definitions do *not* count as prototypes.

Table 30-2 Enforcement levels for Strict Prototype Enforcement

Note

There is one exception to the Require prototypes requirement. A static function does not need a prototype. Just define it with a new-style function definition before using it.

Compiler Settings page

The Compiler Settings page lets you control how THINK C compiles your code.

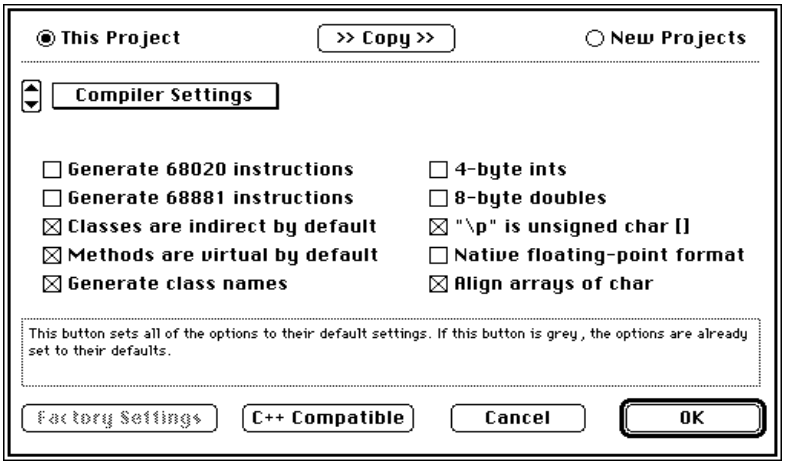

Figure 30-14 Compiler Settings page

This is the only page with the C++ Compatible button. Clicking on this button makes your code compatible with Symantec C++ by turning on the 4-byte ints and Native floating-point format options.

Generate 68020 instructions
If this option is on, THINK C generates code that is optimized for Macintosh computers with the MC68020, MC68030, or MC68040 (such as Macintosh LC, II, IIfx, or Quadra), and that will not run on Macintosh computers with the MC68000 (like Macintosh Classic or Plus). If it's off, THINK C generates code that runs on all Macintosh computers. The factory setting is off.

If the option is on, THINK C generates MC68020 instructions for bit-field operations, addressing, and long word multiplication, division, and modulo operations. Also, the inline assembler accepts all MC68020 instructions and addressing modes for assembly in an `asm { … }` construct.

Note

Before your application uses MC68020 instructions, make sure a MC68020, MC68030, or MC68040 is available. Otherwise, your program may crash. Use the Gestalt Manager, described in the chapter "Gestalt Manager," in *Inside Macintosh: More Macintosh Toolbox*.

Generate 68881 instructions

If this option is on, THINK C generates code that is optimized for Macintosh computers with the MC68881 or MC68882 floating-point unit (such as Macintosh II or IIfx) or the MC68040, and that will not run on Macintosh computers without it (like Macintosh Classic, LC, or Plus). If it's off, THINK C generates code that runs on all Macintosh computers. The factory setting is off.

If this option is on, THINK C generates code for the floating-point coprocessor. You may declare up to five local floating-point variables as `register` variables, and THINK Project Manager places them into MC68881 registers.

If both this option and the Native floating-point format option are on, `long double` types use the MC68881's 96-bit format. Because SANE expects 80-bit `double` types, you'll need to convert from one format to the other. The `float` (32-bit) and `short double` (64-bit) types are identical for SANE and for the MC68881.

Note

Before your program uses MC68881 instructions, make sure a MC68881, MC68882, or MC68040 is available. Otherwise your program may crash. Use the Gestalt Manager, described in the chapter "Gestalt Manager," in *Inside Macintosh: More Macintosh Toolbox*.

Classes are indirect by default

Indirect classes are implemented as handles. Direct classes are implemented as pointers.

If this option is on, classes declared with the `class` keyword are indirect by default. If this option is off, they are direct by default. This option is available only if the THINK C + Objects option is on. The factory setting is on.

This option doesn't affect classes declared with `struct`. A root class declared with `struct` must use the keywords `direct` or `indirect`.

No matter how this option is set, you can still use the words `direct` and `indirect` to declare explicitly how to allocate a class.

For more information on whether to declare classes direct or indirect, see Chapter 23, "Using Objects in THINK C."

Methods are virtual by default
With this option on, all methods are virtual, even those that aren't declared `virtual`. With this option off, only methods declared `virtual` are virtual. This option is available only if the THINK C + Objects option is on. The factory setting is on.

This option doesn't affect all methods. Some are always virtual; others are never virtual:

- Destructors are *always* virtual.
- Constructors are *never* virtual.
- Static methods, including allocators and deallocators, are *never* virtual.

If you're using code you wrote with version 4 or earlier of THINK C, either turn this option on or declare your methods `virtual`. The previous version didn't use the `virtual` keyword and made all methods virtual.

If you're writing new code, turn this option off and declare your methods `virtual` as necessary. Your code will be more compatible with C++.

Generate class names
With this option on, THINK C adds information to class objects to support instantiation by name. With this option off, the compiler won't generate class names, so objects will take up less global space.

4-byte ints
When this option is on, integers are the same size as long integers (4 bytes). When it's off, they're the same size as short integers (2 bytes). The factory setting is off. The C++ compatible setting is on.

Use 4-byte integers if you are:

- Writing a project that mixes C and C++ source files. Symantec C++ does not support 2-byte integers.
- Porting code from a compiler that uses 4-byte integers, like MPW C.
- Porting code that assumes integers and pointers are the same size.

Otherwise, use 2-byte integers, because the MC68000 family handles them more efficiently.

Note

No matter how this option is set, long integers are always 4 bytes long and short integers are always 2 bytes long.

8-byte doubles
With this option on, doubles are the same size as short doubles (8 bytes). When it's off, they're the same size as long doubles. The factory setting is off.

Use 8-byte doubles if you're porting code from a compiler that uses 8-byte doubles, like MPW C. Otherwise, leave the option off, and THINK C uses the most efficient size possible.

When this option is off, the size of doubles is controlled by the Generate 68881 instructions option and the Native floating-point format option.

Note

If both the 8-byte doubles option is on and the Language Extensions option is set to THINK C, no type in THINK C gives you 8-byte floating-point numbers.

"\p" is unsigned char []
When this option is on, Pascal string literals are of type `unsigned char[]`. Otherwise, they're of type `char[]`. The factory setting is on.

Pascal string literals are string literals that start with `\p`, like this:

```
"\pUntitled"
```

If this option is on, Pascal string literals are compatible with the Macintosh Toolbox types `Str255` and `StringPtr`. When you assign a Pascal string literal to a variable of type `Str255` or `StringPtr`, you don't need to cast the string.

Previous versions of THINK C assumed Pascal string literals to be of type `char[]`. Turn this option off only if you have code that takes advantage of this assumption.

Native floating-point format
When this option is off, THINK C uses the universal floating-point format for long doubles. When it's on, THINK C uses either the native SANE or MC68881 floating-point format. The factory setting of this option is off. The C++-compatible setting is on.

Note

Symantec C++ does not support the universal format. If your program mixes C and C++ source files, turn this option on.

The universal format is a floating-point format you can use regardless of the setting of the Generate 68881 instructions option. THINK C automatically converts numbers to and from the universal format. The conversions are too quick to notice unless your applications are computation intensive.

If this option is on, THINK C uses a native floating-point format for long doubles. Which native format it uses depends on the setting of the Generate 68881 instructions option, as shown in this chart:

If Native floating-point is…	And Generate 68881 is…	Then long doubles use this format…
Off	Off	Universal
Off	On	Universal
On	Off	Native SANE
On	On	Native MC68881

Table 30-3 How THINK C chooses a floating-point format

Note

Usually, doubles use the same format as long doubles. However, doubles use the same format as short doubles if the 8-byte doubles option is on.

If your project contains any libraries or projects that were compiled in the universal format, turn this option off. These libraries use the universal format:

- ANSI
- ANSI-small
- ANSI-A4
- Profile

If you're writing a library for general use, turn off both the Native floating-point format option and the Generate 68881 instructions option. Projects can use your library no matter how they set the Generate 68881 instructions option, as long as they turn off the Native floating-point format option.

If you want the fastest code possible and use no libraries compiled in the universal format, turn the Native floating-point format option on. The conversions to and from the universal format are eliminated.

Align arrays of char
If this option is on, THINK C will always align arrays of type `char` to word (two byte) boundaries. If this option is off, THINK C may place an array of type `char` at a nonaligned address.

Code Optimization page

The Code Optimization page lets you control how THINK C optimizes your code.

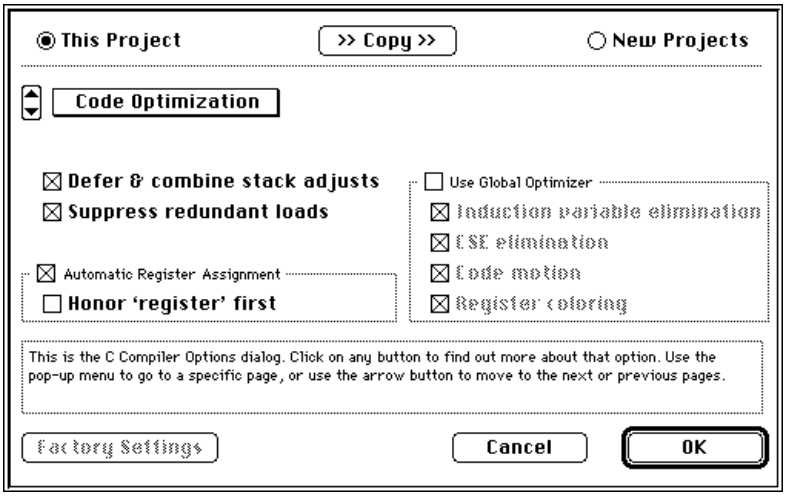

Figure 30-15 Code Optimization page

The options on this page fall into two categories. The options in the left column control optimizations that the compiler performs. Using them doesn't increase your compile time significantly.

The options in the right column control the global optimizer, which adds an additional pass over your compiled code. Using the optimizer may improve your code, but your compilation time could double.

Defer & combine stack adjusts
With this option on, THINK C saves time and space in code that calls many functions in a row.

With this option off, THINK C follows this function-calling rule: When a statement calls a C function, that statement pushes the arguments on the stack before calling the function and pops them off after it returns.

With this option on, THINK C changes that rule. If your code contains a run of C functions, it doesn't pop a function's arguments off the stack immediately after one of the function returns. It continues to call the other functions and push their arguments on the stack. It lets the arguments accumulate and keeps track of how much

is on the stack. When the run of functions is done, it pops the arguments off all at once. The factory setting is on.

Suppress redundant loads
If this option is on, THINK C doesn't load data into a register when that data is already in a register. This optimization makes your code smaller and faster.

Automatic register assignment
If this option is on, THINK C automatically assigns variables to registers to make your code as small as possible. With the Honor 'register' first option, you can choose whether THINK C also puts the variables declared `register` into registers. If this option is off, THINK C puts only those variables declared `register` into registers.

If you select this option, the Honor 'register' first option appears. The two settings for this option are described in the following table:

If Honor 'register' first is…	THINK C…
On	Puts the variables declared `register` in registers and puts the variables it wants in the remaining registers.
Off	Ignores your `register` declarations and puts the variables it wants in registers.

Table 30-4 The Honor 'register' first option

Use global optimizer
The option controlling the global optimizer is a "master switch." If this option is off, the options below it are disabled. If it's on, the options below it are enabled, set to the values they had the last time the master switch was on.

You probably won't use the global optimizer while you're debugging. It adds an additional pass over your compiled code and may double your compilation time. Also, it generates machine code that can be significantly different from your source code. The debugger may not be able to pick out the machine code instructions that correspond to a statement.

The following four options become available if the Global Optimizer option is enabled:

Induction variable elimination
This optimization makes loops faster, especially those that cycle through an array. It will also make your code slightly larger. Use this optimization if your code contains many loops. It's especially useful on a Macintosh with a MC68000, like a Macintosh Classic or Plus, which performs 32-bit multiplication in software.

CSE elimination
This optimization makes your code smaller and faster. It replaces subexpressions that are used more than once with a temporary variable set to the subexpression's value. Use this optimization on all your code. It's especially useful if you also use the Register coloring option.

Code motion
This optimization makes your loops faster but may make your code slightly larger. It moves expressions that remain constant in each iteration out of loops. Use this optimization if your code contains many loops.

Register coloring
This optimization makes your code smaller and faster. It searches your code for variables of the same type that are never used at the same time. It treats them as if they were the same variable. If there's a register free, it lets them share that register.

Debugging page

The Debugging page lets you control how THINK C generates stack frames, profiler calls, and Macs-Bug names.

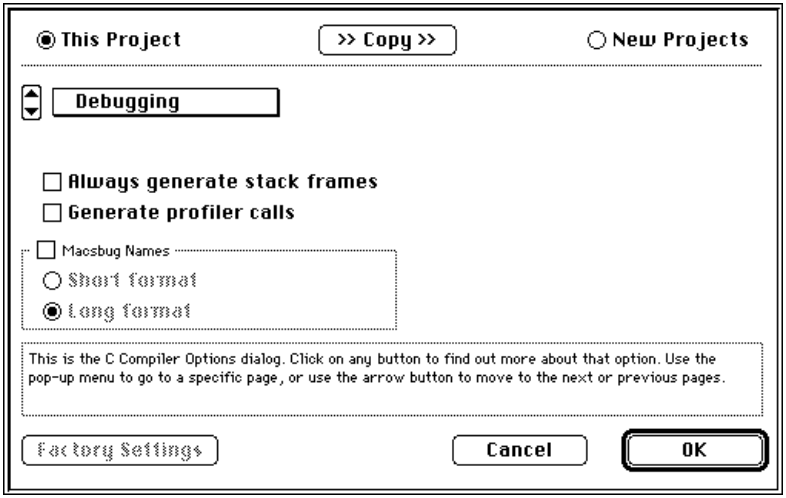

Figure 30-16 Debugging page

Always generate stack frames
When this option is on, THINK Project Manager generates a stack frame for almost every function called. When this option is off, THINK Project Manager doesn't generate a stack frame for functions unless they have local variables or parameters. The factory setting is off.

If you're using the debugger's call chain menu or the profiler, turn this option on. Otherwise, leave it off. Your program will be smaller and faster without the unnecessary stack frames.

Generate profiler calls
When this option is on, THINK C generates calls to code profiler routines. The code profiler collects timing statistics about your functions. The factory setting is off.

THINK C can profile only functions that have stack frames. To create stack frames for almost every function, turn on the Always generate stack frames option.

See Chapter 25, "Profiling Projects," to learn more about the code profiler.

MacsBug names
This option lets you choose whether THINK C includes function names in your compiled code for assembly language level debuggers such as MacsBug or TMON. If this option is on, you can choose between two types of names, Short format and Long format. If this option is off, THINK C doesn't include function names in your compiled code.

THINK C generates symbols only for functions that have stack frames. To create stack frames for almost every function, turn on the Always generate stack frames option described in "Always generate stack frames" earlier in this chapter. Be aware that while MacsBug symbols are useful for debugging, they add at least 8 bytes to every procedure.

If you use Short format names, function names are limited to eight characters. Underscores are stripped off. Names shorter than eight characters are padded with spaces. For methods (such as `TDrawWindow::FileWrite`), only the first eight characters of the class and method are stored (such as `TDrawWin::FileWrit`). The THINK Debugger, TMON, and all versions of MacsBug can use these names.

If you use long format names, function names can be any length. For methods (such as `TDrawWindow::FileWrite`), the full class and method names are stored (such as `TDrawWindow::FileWrite`). The THINK Debugger, TMON, and MacsBug 6.0 and later can use these names.

Prefix page

The Prefix page lets you write preprocessor code that THINK C will include in all your files.

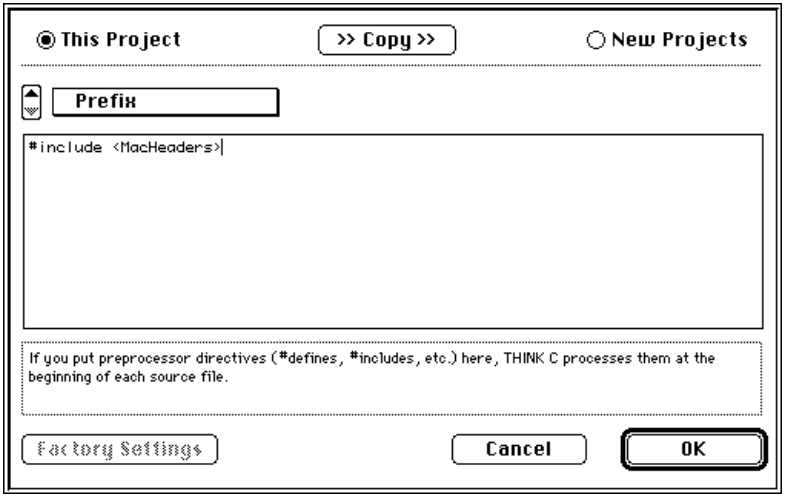

Figure 30-17 Prefix page

This page lets you include some text in all the C source files in your project. It's as if you put the code from this page at the beginning of all your C source files.

If you use a precompiled header file, like `MacHeaders`, include it here. By default, this page contains the line:

```
#include <MacHeaders>
```

If you need to define a macro in all your files, define it here. For example, you may have some debugging code in your files that's compiled only if the macro `DEBUG` is defined. To include that code, include this line here:

```
#define DEBUG
```

When you don't need to include the debugging code anymore, just delete that line from this page. You don't need to edit every C source file in your project.

THINK Rez options

The **THINK Rez options** dialog lets you choose options regarding resource alignment, prefix strings, and redeclared types.

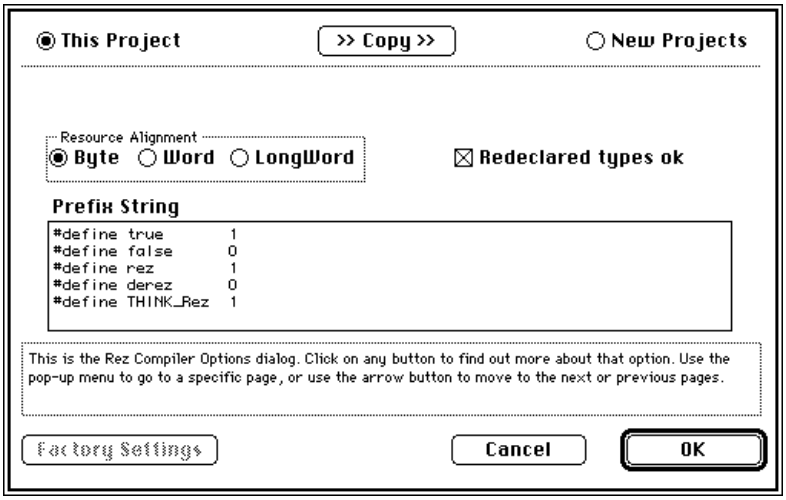

Figure 30-18 THINK Rez options

Resource alignment
You use this option to indicate how you want your resources aligned: along byte, word, or long word boundaries. The default alignment is byte.

Redeclared types ok
This option lets you choose if THINK Rez continues compiling your file if you declare a resource type more than once in a resource file. When it's on, THINK Rez prints a warning and continues compiling your file. Otherwise, THINK Rez prints an error message and stops compiling. The default for this option is on.

Prefix string
The prefix string field lets you include some text in all the THINK Rez source files in your project. The effect is the same as if you put the string at the beginning of all your resource description files. By default, the option defines the macro `true` to be `1`, `false` to be `0`, `rez` to be `1`, `derez` to be `0`, and `THINK_Rez` to be `1`.

If you need to define a macro in all your files, define it here. For example, you may have some resources you use for debugging that

are compiled only if the macro DEBUG is defined. To include that code, include this line here:

```
#define DEBUG
```

When you don't need to include the debugging code anymore, just delete that line from this page. You don't need to edit every .r file in your project.

Note

When you change the THINK Rez Prefix String, the THINK Project Manager recompiles all the THINK Rez files in your project.

Set tabs & font

Use this command to change the tab stops and the font used by the THINK C editor. You can select different tab stops and fonts for each Edit window. When you choose this command, you see a dialog box like the one in Figure 30-19.

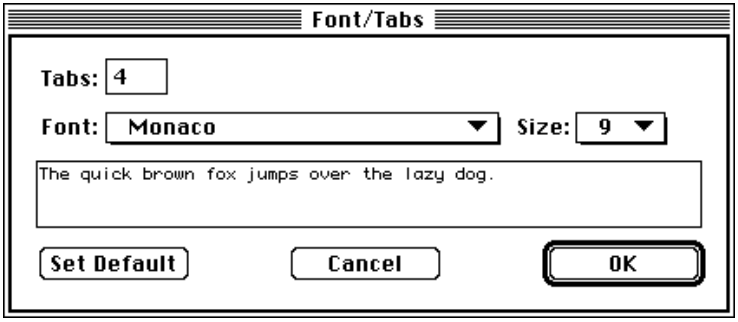

Figure 30-19 Set Tabs & Font dialog

Type a number to set the number of spaces per tab. To change the font, click the font name. A pop-up menu opens with the names of the fonts in your System file. Click the font size to see a pop-up menu of possible font sizes.

Click OK to use these font and tab settings for the current file only. Click the Set Default button to use these settings as the defaults for all files that don't have font and tab information. Click Cancel if you decide not to make a change.

If you are using a proportionally spaced font like New York or Geneva, the THINK C editor uses the width of the letter “m” to figure out the width of a tab.

When you change font or tab settings, the editor adds `'EFNT'` and `'ETAB'` resources to your text files to record the new settings. Other text editors use these resources as well.

Shift Left

This command shifts a selected range of lines to the left. It deletes the first character of each line in the selected range, if that line begins with a tab.

Shift Right

This command shifts a selected range of lines to the right. It inserts a tab at the start of each line in the selected range.

Balance

This command extends the current selection in both directions until it encloses the smallest surrounding balanced text enclosed in parentheses `()`, brackets `[]`, or braces `{}`. Successive invocations select larger sequences of text.

This is a quick way to check whether all your function definitions are properly balanced. Start at the beginning of a file and search for the first left brace (“`{`”). Then use **Balance** and **Find Again** commands repeatedly until you get to the end of the file.

Balance is a textual operation. The search does not exclude comments or strings. If you have a lone brace, bracket, or parentheses in a comment, **Balance** will try to find a match for it.

The Search Menu

*The commands in the **Search** menu let you find and replace strings in your source files. THINK Project Manager has extensive search and replace functions, including multi-file searching and pattern-matching searching.*

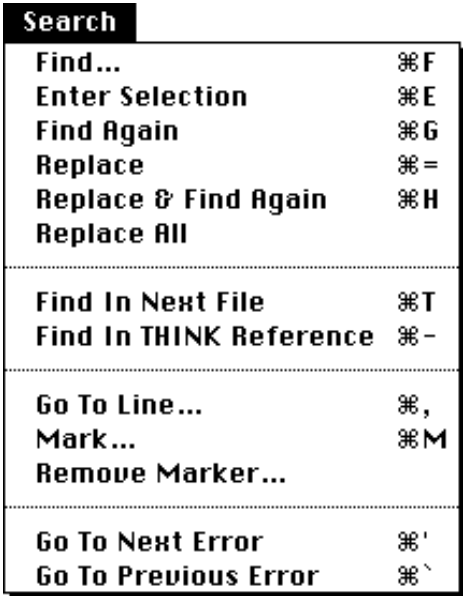

Figure 30-20 Search menu

Find

This command lets you specify a string to search for. If the string is found, it is highlighted. If it is not found, the editor simply beeps.

When you choose **Find**, a dialog appears on which you specify the search string, as well as an optional replacement string. You can type the search and replacements strings in the edit boxes, or choose

from the last five strings you have entered from pop-up menus. To use the menus, click the arrows to the right of the edit boxes.

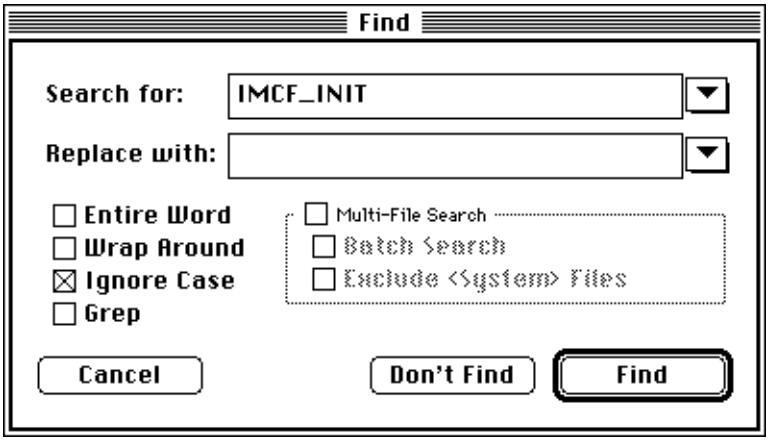

Figure 30-21 Find dialog

You can also set search characteristics in this dialog box. Note that you can set these same options on the Editor page of the **THINK Project Manager options** dialog. When you set the options in the **Find** dialog, though, they apply only to the current session. If you want to set the defaults permanently for the current project or for all new projects, use the **THINK Project Manager options** dialog.

Option	Description
Entire Word	When this option is checked, the editor's search and replace functions work on whole words only. For example, a search for "string" will not match the word "strings". This option is off by default.
Wrap Around	With this option on, the editor's search and replace functions search the entire file rather than from the current position to the end of the file only. At the end of the file, the search "wraps around" to the beginning of the file and continues. This option is off by default.

Table 30-5 Find dialog search characteristics

Option	Description
Ignore Case	When this option is checked, the editor's search and replace functions disregard case when performing a search. A search string matches either upper- or lowercase. This option is on by default.
Grep	With this option on, the editor's search and replace functions perform searches using a powerful pattern-searching feature based on the `grep` command found in many Unix systems. This option is off by default.
Batch Search	With this option on, the editor's multi-file search function finds all matching occurrences of the search pattern and displays them in a window, instead of stopping at the first matching occurrence. This option is off by default.
Exclude <System> Files	With this option on, the editor's multi-file search function does not search through files that are in the THINK Project Manager tree.

Table 30-5 Find dialog search characteristics *(Continued)*

Click the Find button to go ahead with the search or the Cancel button to cancel of out the operation. Click the Don't Find button to have the editor accept the new string and option settings but not initiate a search. This is a useful option if you want to set values for a replace operation without executing the first find.

Enter Selection

This command sets the search string to the current selection, clearing Grep and Multi-File Search. You can then use **Find Again** to begin searching, or **Find** to set search options.

Find Again

This command searches for the next occurrence of a previously specified string.

Replace

This command replaces the current selection with a replacement string. If you haven't provided a replacement string, the string that has been found is cleared and replaced with nothing. Usually, you use this command after finding a string.

Replace & Find Again

This command replaces the current selection with the replacement string. It then finds the next instance of the search string but does not replace that string. Use this command to step through a series of replacements. After each replacement, you see the next instance of the search string, so you can decide whether to replace it. If you do, use the **Replace** or **Replace & Find Again** commands. If not, use the **Find Again** command to find the next occurrence of the string.

If you haven't provided a replacement string, this command clears the string that has been found and replaces it with nothing.

Replace All

This command replaces every instance of the search string. If the Wrap Around option is on, it replaces every instance in the file. If the Wrap Around option is off, it replaces every instance from the current cursor position to the end of the file. Use this command when you don't want to give your approval for every replacement. If you haven't provided a replacement string, this command will clear the string that has been found and replace it with nothing.

Warning

You cannot undo this command.

Find in Next File

This command lets you search for a string through more than one file.

To use this command, you must click the Multi-File Search check box in the **Find** dialog box. When you check this box, another dialog box displays all of the text files known to THINK Project Manager. You can scroll through the list and select files by clicking them to place check marks by the names. You can also use the buttons in the dialog box to Check All, Check None, Check Sources, or Check Includes. (If a file is already selected, clicking its name will remove the check mark.)

When you've checked the files you want to include in the multi-file search, click OK to return to the **Find** dialog box.

THINK Project Manager searches for the string specified in the **Find** dialog box through each of the files that have been checked, starting with the first one. If the search string is found in a given file, THINK Project Manager opens an edit window containing the file, and the search string is selected. At this point, you can make any edits you choose. If you want to search further in the current file, use the **Find**, **Find Again**, **Replace**, and **Replace All** commands, which

work within the current file. When you're ready to go on with the multi-file search, use the **Find in Next File** command.

Multi-file search is useful when you are writing a program and you decide to modify a function that is used in multiple files. You can open each of the files containing the search string, so you can switch back and forth between the various edit windows as necessary. (This feature is also helpful when you are tracking down link errors due to undefined or multiply defined symbols.)

Find in THINK Reference

If you're running System 7, you can look up information in THINK Reference from the THINK Project Manager. If you select a function, type, or variable name described in one of the libraries and choose this command, the THINK Project Manager launches THINK Reference (if it's not already open), which opens to the page that describes the symbol. If THINK Reference is already running and you Option-double-click on a symbol, the THINK Project Manager switches to THINK Reference, which opens to the page that describes the symbol. To return to the THINK Project Manager, click the Iconize button in the THINK Reference window.

Note

THINK Reference 1.0 does not support this feature. You must have THINK Reference 2.0.

If you don't want the THINK Project Manager to switch to THINK Reference when you look up the definition of a symbol, hold down both the Control and Option keys when you double-click the symbol.

Go To Line

This command lets you move to a specific line in your file. You need to know the line's number. Lines are numbered consecutively, with the first line in the file being numbered 1.

When you choose **Go To Line**, the dialog in Figure 30-22 opens.

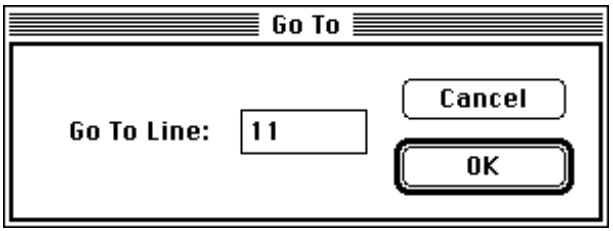

Figure 30-22 Go To Line dialog

The default line is the line that the insertion point is in. Type a different number and click OK. If you change your mind, click Cancel. The editor moves the insertion point to the beginning of the line with the number you typed.

Mark

This command lets you place a marker in a file. Place the insertion point in the line you want to mark or select some text in that line. Choose this command from the **Search** menu. The dialog in Figure 30-23 appears.

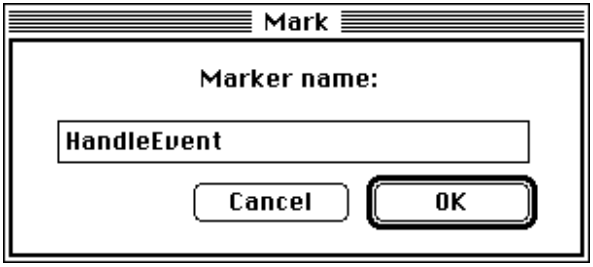

Figure 30-23 Mark dialog

Edit the marker name if you want, and click OK. If you selected text, the editor uses that text as the default marker name. Otherwise, the editor uses the word nearest the insertion point. Click Cancel to back out of the operation.

To go to a marker in a file, hold down the Command key and click the title bar of the file's edit window. A pop-up menu of the file's markers appears, sorted alphabetically.

Remove Marker

This command lets you delete a marker from a file. The dialog in Figure 30-24 appears.

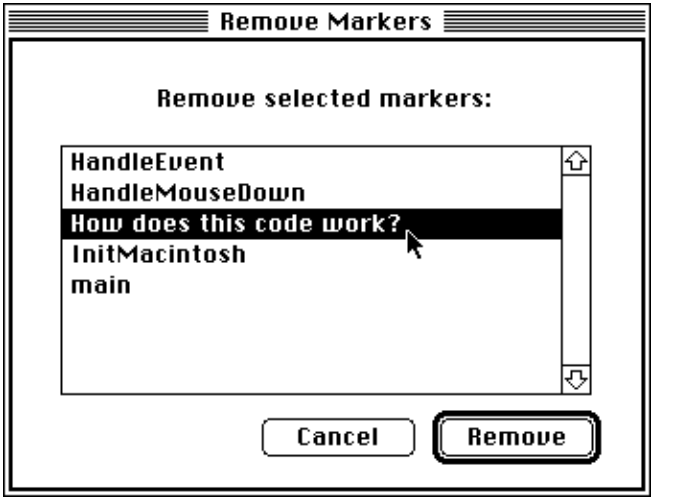

Figure 30-24 Remove Marker dialog

Select the markers you want to delete and click OK. If you change your mind, click Cancel. The next time you see the marker pop-up menu, the deleted markers will be gone.

Go To Next Error

Use this command to go to the next error.

Go To Previous Error

Use this command to go to the previous error.

The Project Menu

*The commands in the **Project** menu work with the current project and with other projects already created. You can open projects, set the project type, and make sure all the files in the project are compiled and loaded. This menu also contains commands for building a file containing your application, desk accessory, device driver, or code resource.*

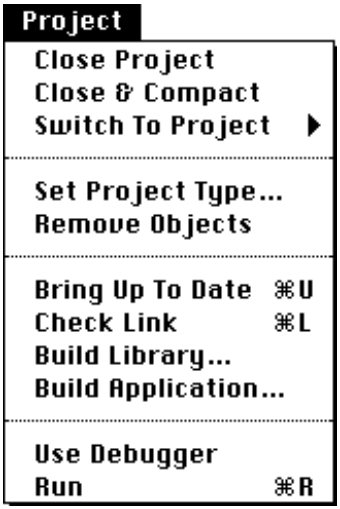

Figure 30-25 Project menu

Close Project

This command closes the current project and then lets you open an existing project or create a new project. If you try to close a project with open files that have been changed but not saved, a dialog box prompts you to save them. To disable this dialog, turn the Confirm project updates option off on the Preferences page under the **Edit** menu by choosing **Options** and then **THINK Project Manager**. When this option is off, changed files will automatically be saved when you close.

Close & Compact

This command has the same effect as **Close Project**, but it makes the project document as small as possible without removing any object code. Use this command before you back up your project, when you plan to use a project as a library, or when you plan to transmit the project through a modem. (See also the **Remove Objects** command.)

Switch to Project

This command accesses a hierarchical menu from which you open an existing project. The menu contains all projects you've opened in the current session and all projects that have aliases in the `Projects` folder, which is in the folder that contains the THINK Project Manager.

Set Project Type

This command lets you set the project type. The default project type is Application, but you can change it to Desk Accessory, Device

Driver, or Code Resource. All project types let you specify the File Type and Creator of the file that is created by one of the **Build** commands. To learn the details of each project type, see Chapter 13, "Building Projects."

Set the project type before compiling any of your sources. THINK Project Manager will need to throw away any existing object code if you switch the project type once there is compiled code in the project. If the project already contains files, THINK Project Manager asks if you're sure you want to change the project type.

The **Set Project Type** dialog box lets you specify different attributes for each type of project.

Application dialog

The Application dialog lets you specify how string literals and floating-point constants are stored and lets you set some of the fields of the SIZE *resource MultiFinder uses.*

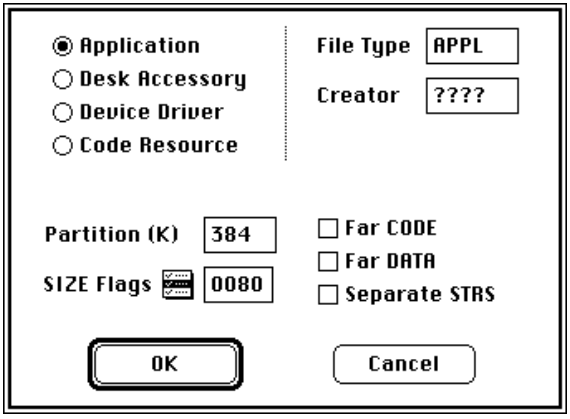

Figure 30-26 Application dialog

Partition
The value in this field is the amount of memory the system MultiFinder allocates for your application. The default is 384K, which is more than enough for most moderately sized applications.

SIZE flags
The pop-up menu lets you set the bits that tell the system which system services your application uses or is compatible with. You can use the pop-up menu or type a hex value into the field.

Far CODE
If you need a large jump table, check the Far CODE option. It allows a jump table as large as 256K, since it uses 32-bit addresses instead

of 16-bit. Your application will be about 6% larger because of the longer addresses. If the Far CODE option is off, your jump table can only be as large as 32K.

Note

If the Far CODE option is on, each of your project's files can contribute only 4095 jump table entries.

Far DATA

If you use a lot of global data, check the Far DATA option. It allows you to have an unlimited number of global variables, since it uses 32-bit addresses instead of 16-bit. Your application will be about 8% larger because of the longer addresses. If the Far DATA option is off, your project can contain only 32K of global variables.

Note

If the "Far DATA" option is on, each of your project's files can contribute only 32K of global data. This limit means, for example, that you can't declare a static array that's over 32K.

Separate STRS

When this option is on, string literals and floating-point constants are placed in their own STRS component. Otherwise, they are part of the DATA component.

Desk Accessory and Device Driver dialogs

The Desk Accessory and Device Driver dialogs are similar. For desk accessories, the File Type and Creator fields are filled in for you so the resulting file will be a Font/DA Mover file. There are other differences between Desk Accessories and Device Drivers that have to do with the header fields.

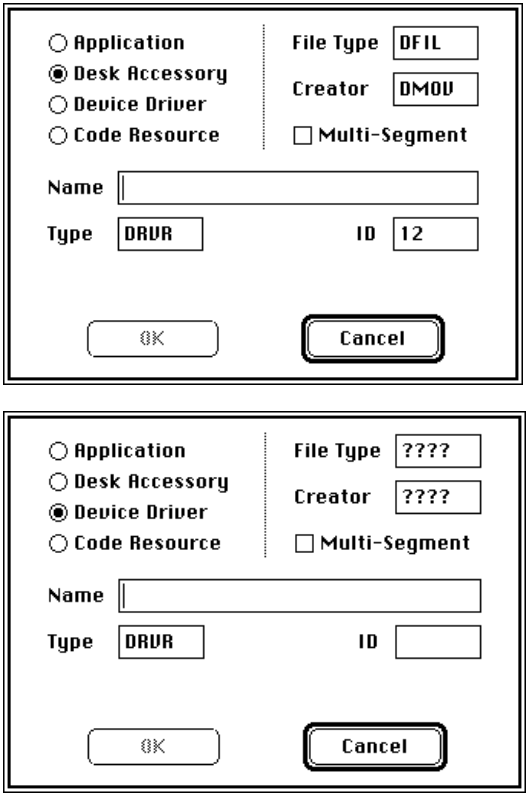

Figure 30-27 Desk Accessory and Device Driver dialogs

Multi-segment
When this option is on, your desk accessory or device driver can have up to 30 segments.

Name
This field is the name of your desk accessory or device driver. By convention, desk accessory names begin with a null byte. THINK Project Manager takes care of this for you. Device drivers begin with a period. If you don't provide one, THINK Project Manager will add one for you.

Type
Desk accessories and device drivers are resources of type DRVR. You can change the type if necessary.

ID
The ID number refers to the `'DRVR'` resource. Desk accessories default to `12`. The Font/DA Mover will renumber your desk accessory (and its owned resources) if there is a conflict.

Code Resource dialog

The Code Resource dialog lets you specify whether your code resource will begin with a standard header, its name, type, and ID, and its resource attributes.

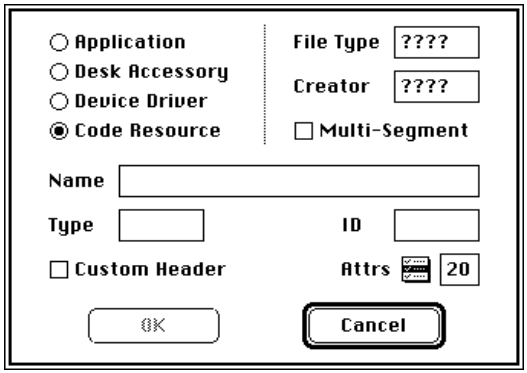

Figure 30-28 Code Resource dialog

Multi-segment
When this option is on, your code resource can have up to 30 segments.

Name
Use this field for the name of your code resource. For most code resources, the name is optional.

Type
This field contains the resource type of your code resource.

ID
This field contains the ID number of your code resource.

Custom header
When this option is off, THINK Project Manager uses a standard header for your code resource. The header places the address of your resource in register `A0` and branches to your `main()` function. If you check this option, your code resource will begin with the first function in the file in which `main()` is defined.

Attrs
This field contains the resource attributes (locked, purgeable, system, and so on) of your code resource. You can select the attributes from the pop-up menu, or you can set them by typing a hex number into the field.

Remove Objects

This command removes all the object code from a project. It reverses the effects of all previous compilations and loading of libraries. The project document is "dehydrated"; it can be "reconstituted" by recompiling all source files and reloading all libraries.

Use **Remove Objects** when you need to make the project document as small as possible—for example, for archiving or for transmitting to someone else.

Bring Up To Date

This command compiles all source files that need to be compiled and loads any libraries that haven't been loaded.

Check Link

This command checks for the same error conditions as **Run** or **Build Application** does (except segment overflow), but without running the project or building an application. If any files need to be compiled or loaded, you will be prompted to bring the project up to date, even if you have set **Confirm Auto-Make** off.

The **Check Link** command displays errors in the **Link Errors** window. If there are multiply defined or undefined symbols, the names of the files containing the symbols appear in parentheses.

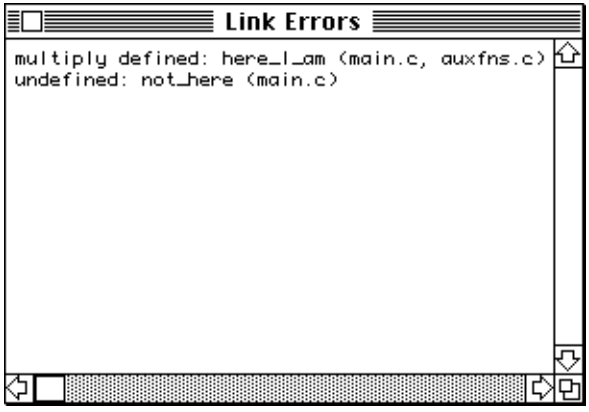

Figure 30-29 Link Errors window

When the **Link Errors** window comes up as the result of an attempt to run or build, this additional information is not displayed. Since some disk activity is required to compute the information, the window is displayed only when you specifically request **Check Link**.

Build Library

This command saves the current project as a single binary file that can be added as a library to other project documents. A dialog box prompts you for the name of the library file. The convention is `name.Lib`; however, a library may have any valid file name. Note that you can include a project in another project without first saving it explicitly as a library.

Build Application
Build Desk Accessory
Build Device Driver
Build Code Resource

These commands save the current project as an application, desk accessory, device driver, or code resource. A dialog box lets you name the resulting file (the dialog on the left is for applications, desk accessories, and drivers; the one on the right is for code resources):

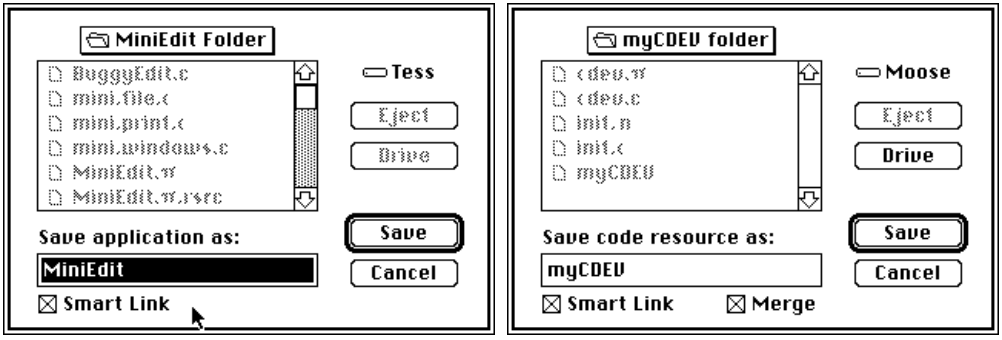

Figure 30-30 Build dialog

If the Smart Link option is checked, THINK Project Manager uses the smallest number of code components of the source files or libraries to create the resulting file. It takes a little longer to put the application or resource together, but the resulting file will be as small as possible. Do not use this option if you're building frequently for testing.

If the Merge option is checked in the **Build Code Resource** dialog, you can place your code resource into an already-existing file. Just type the name of the file.

If there is a file with the same name as the project that ends in `.rsrc` in the same folder as the project, the resources in the `.rsrc`

file will be merged into the resulting file. However, code resources built with the Merge option will not use the `.rsrc` file.

Use Debugger

This command turns the source debugger on and off. When the source debugger is on, you'll see a "bug" column in the project window and, when you run the project, the debugger's windows appear on the screen. Selecting this command is the same as clicking on the Use Debugger check box on the Debugging page of the **Options** dialog box. See Chapter 12, "Debugging Programs," to learn how to use the source-level debugger.

Run

This command runs the program contained in the project. If the project is not up to date, a dialog box will prompt you to bring it up to date. If the Confirm Auto-Make option on the Preferences page of the **THINK Project Manager options** dialog is not checked, then the project will automatically be brought up to date. This lets you edit a source file and run the changed program, without ever explicitly recompiling or relinking the program.

If the project type is Desk Accessory, the **Run** command uses an auxiliary program, DAShell, to run your desk accessory. The THINK Project Manager needs to build you desk accessory and save it in a file first before the THINK Project Manager launches DAShell. Your desk accessory will be in the **Apple** menu.

The Source Menu

*The commands in the **Source** menu let you add source files or remove them from your project. This menu also contains commands to create precompiled headers, to compile source files, to load libraries, and to control the automake facility.*

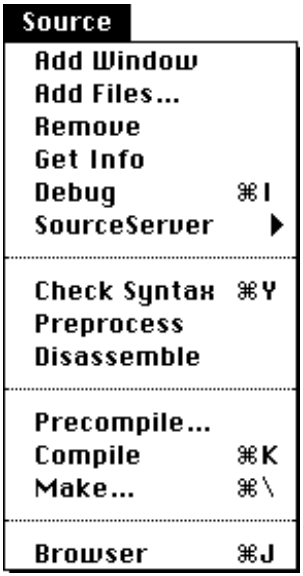

Figure 30-31 Source menu

Add Window

This command adds the file in the frontmost edit window to the project to the Project window. The file must end in one of the extensions defined on the Extensions page of the **THINK Project Manager options** dialog.

Add Files

This command lets you add existing source files and libraries to a project. It brings up the dialog in Figure 30-32.

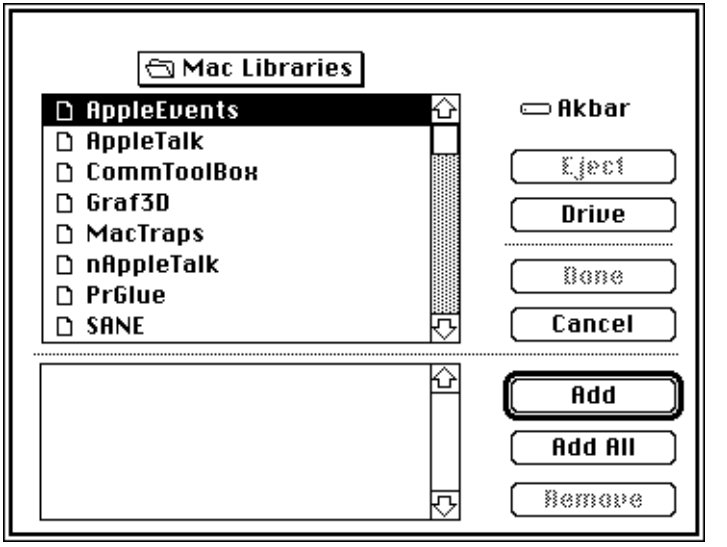

Figure 30-32 Add Files dialog

The top list displays the contents of the current folder. The bottom list displays the files that will be added to your project when you click Done. You add files to the bottom list with these commands:

- To add a file, select it and click Add, or double-click it.
- To add all the files from the current folder, click Add All.
- To open a folder, select it and click Open, or double-click it. (The Add button becomes the Open button when you click on a folder.)
- To remove a file from the bottom list, select it and click Remove.
- When you finish selecting files to add, click Done.
- If you change your mind and don't want to add any files, click Cancel.

Remove

This command removes a selected source file or library from the project.

Get Info

The **Get Info** command displays a dialog box that contains the sizes of each project component. It initially displays information for the currently selected file.

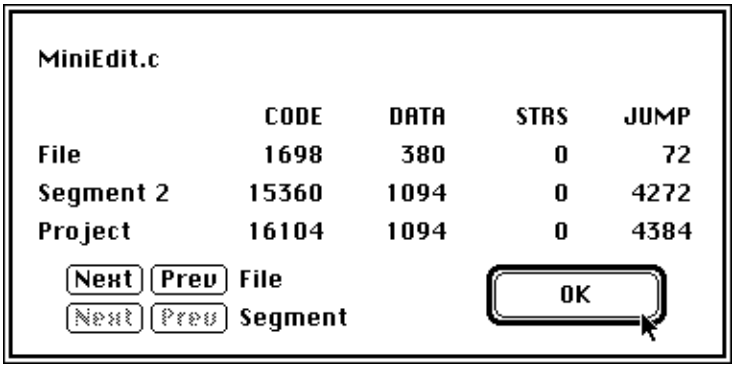

Figure 30-33 Get Info dialog

The dialog shows the size of the CODE, DATA, STRS, and JUMP components for the selected file. If the file hasn't been compiled, these values will be 0 (zero). The dialog also shows the same information for the segment and the entire project. The Next and Prev buttons let you examine the other files and segments in the project.

Debug

When you're using the source-level debugger, this command sends the frontmost edit window (or the selected file in the Project window) to the Source window of the debugger.

SourceServer

SourceServer lets you handle source and version control issues, such as those that occur when you work on a shared project or one with multiple sources.

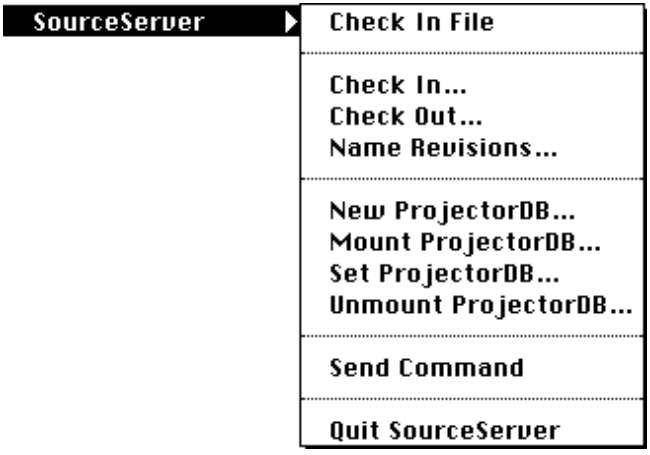

Figure 30-34 SourceServer hierarchical menu

To use SourceServer, choose **Launch SourceServer** from the **SourceServer** menu. This launches SourceServer and initializes it for use with THINK Project Manager. These commands are described in detail in Chapter 24, “Using SourceServer.”

Command	Description
Check In File Check Out File	Checks in, or checks out, the file in the frontmost edit window
Check In	Displays a dialog that lets you check files into the current SourceServer DB
Check Out	Displays a dialog that lets you check files out of the current SourceServer DB
Name Revisions	Displays a dialog that lets you create a named set of files which you can easily check in or check out all at once
New ProjectorDB	Displays a dialog that lets you create a new SourceServer DB

Table 30-6 The SourceServer commands

Command	Description
Mount ProjectorDB	Displays a dialog that lets you mount a SourceServer DB
Set ProjectorDB	Displays a dialog that lets you choose which of the mounted SourceServer DBs is the current one
Unmount ProjectorDB	Displays a dialog that lets you unmount a SourceServer DB
Send Command	Sends the line in the frontmost edit window to SourceServer and displays the result under the line
Launch SourceServer Quit SourceServer	Launches, or quits, the SourceServer application

Table 30-6 The SourceServer commands *(Continued)*

Check Syntax

This command lets you check the syntax of a file. It checks the front window but does not add the file to the Project window.

Preprocess

This command runs the code in the frontmost window through the translator's preprocessor and displays the result in a new window. It's especially useful if you think you have a bug in one of your macros. The preprocessor expands your macro, includes the contents of your `#include` files, and evaluates your `#ifdef` statements. You can save and print the contents of this window like you would any other file.

Disassemble

This command disassembles the code in the frontmost window and displays the result in a new window. Use this command to figure out your code's efficiency and how to write inline assembly code. You can save and print the contents of this window like you would any other file.

Precompile

This command creates a precompiled header from the contents of the frontmost edit window. Precompiled headers may not have any code or data definitions. You can include `#include` files (even another precompiled header) in precompiled headers.

Compile
Load Library

These commands let you compile a single source file or load a single library without compiling the whole project.

When an edit window is frontmost or when a source file is selected in the Project window, the **Compile** command is available. It compiles the contents of the file and posts the result in the project document. You can compile only files that end in an extension defined on the Extensions page of the **THINK Project Manager options** dialog. To check the syntax of a source file without adding it to the project document, use the **Check Syntax** command instead.

When a library file is selected in the Project window, the **Load Library** command is available. It loads the code for the library into the project. To add a library to a project for the first time, use the **Add** command.

Make

When you select this command, a dialog box appears showing all the files in the project in a scroll box. The files that THINK Project Manager thinks need to be recompiled (or, in the case of libraries, reloaded) are checked. You can alter the list by using the cursor to check or uncheck files, or by clicking the Check All, Check All .c or Check None buttons. Your changes will not be remembered if you click Cancel.

When you click the Make button, THINK Project Manager brings the project up to date. It recompiles files that need to be recompiled and loads libraries that need to be reloaded. If you click the Don't Make button, the project is updated the next time you use any of the

commands that update the project: **Bring Up To Date**, **Run**, **Check Link**, or **Make**.

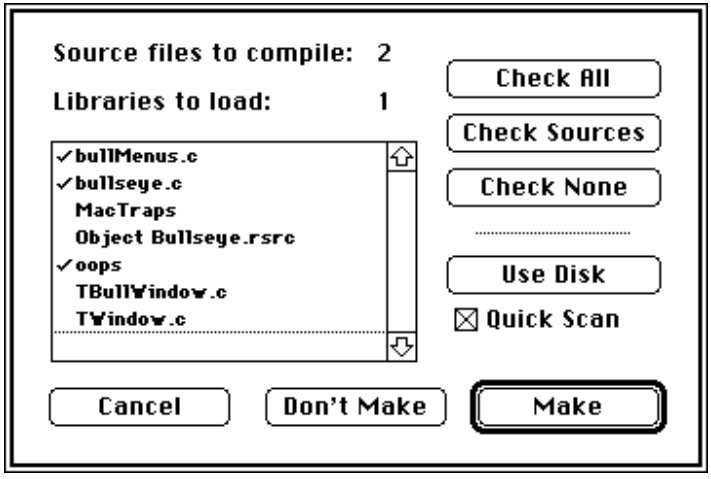

Figure 30-35 Make dialog

If you click the Use Disk button and the Quick Scan check box is checked, THINK Project Manager checks the date/time-modified of all the files in the project. Normally this is not necessary because THINK Project Manager automatically tracks the changes you make as you edit. This knowledge is project-specific, though, so if you have a source file that belongs to two different projects and you change it in one, the other project won't know it's been changed unless you click the Use Disk button.

If the Quick Scan check box is not checked, THINK Project Manager does a more extensive check. If a file can't be found, THINK Project Manager searches the tree the file was originally in. (The Use Disk feature can't help you detect when you've moved files from one tree to another. See "Moving Files on Disk" in "Organizing Files and Folders," Chapter 9, for details.)

Use Disk displays its progress if the Quick Scan option is off. You can abort it with Command-Period, although the next instance of Use Disk will start again at the beginning.

If a source file or library is not found by the Use Disk operation, its status. in other words, whether it is checked in the **Make** box) is unchanged. However, if a source file is found but one of the files it

includes is not found, the source file is marked as needing to be compiled or checked.

Clicking Cancel does not undo the effect of the Use Disk operation. Unlike the other buttons, which simply add (or remove) a check mark to those specified by Auto-Make, Use Disk actually updates the date/time record associated with each file that is in the project. You can, of course, manually check or uncheck files after clicking Use Disk.

Browser

This command invokes the class browser, a tool that displays all the classes defined in your project as a tree chart. You can use the Class Browser to look at the declaration of classes or to find the definitions of methods.

The Windows Menu

*The **Windows** menu has three sections, separated by dotted lines. The first section contains five commands: **Arrange**, **Zoom**, **Full Titles**, **Close All**, and **Save All**. The second section contains an entry for the Project window and one for each untitled window. The third section contains an entry for each file open in an edit window, in alphabetical order.*

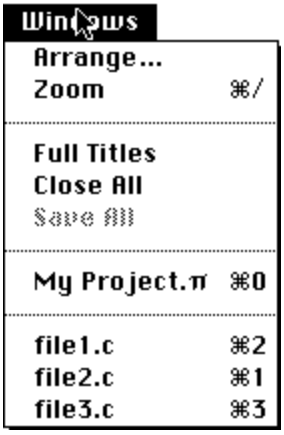

Figure 30-36 Windows menu

Arrange

Use this command to rearrange the editing windows so all their title bars are visible. The dialog in Figure 30-37 appears.

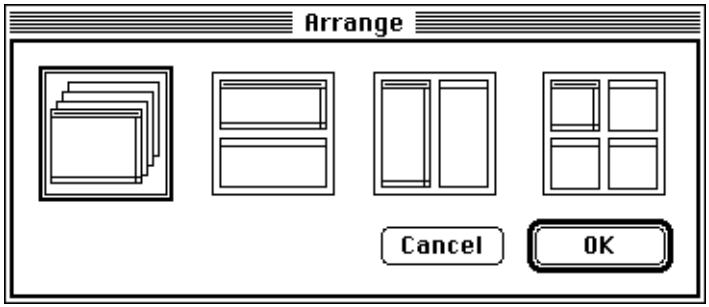

Figure 30-37 Arrange dialog

The four pictures represent different ways to arrange your editing windows, as described in Table 30-7. To rearrange your windows, double-click one of the pictures, or select one of the pictures and

click OK. To leave your windows in their current arrangement, click Cancel.

Picture	Description
	Stack, with the windows staggered. Each window is slightly below and to the right of the window beneath it. Only the title bars of the windows in the back are visible.
	Vertical tile. The front window takes up the top half of the screen. The other windows are stacked in the bottom half.
	Horizontal tile. The front window takes up the left half of the screen. The other windows are stacked in the right half.
	Full tile. The windows are tiled so they're all visible. The front window is in the top-left corner of the screen.

Table 30-7 Arrange options

Zoom Resizes the frontmost window to occupy the full screen. If the window is already at full screen, it is restored to its previous position and size. This command has the same effect as clicking the zoom box in the upper-right corner of the window.

Full Titles When this option is checked, the title of each edit window indicates the full path name of the file. The default is off.

Close All This command closes all the edit windows. If the Confirm Saves option is checked, THINK Project Manager prompts you to save each modified window; otherwise, windows are automatically saved. Holding down the Option key as you click the close box of an edit window has the same effect as using the **Close All** command.

Save All Use this command to save all the modified windows. No confirmation is requested.

Project window The Project window is brought to the front. (The menu item appears as the name of the project, not the words "Project window.")

Edit Windows Use this section of the **Windows** menu to bring the selected window to the front. The number reflects the "slot number," or the initial

position of the window. (The first created window occupies slot #1, and slots #6-10 occupy the same screen positions as slots #1-5, and so on. Slots vacated by closed windows are reused at the next opportunity.) The number of windows is limited only by available memory, but only windows in the first nine slots have a Command-key equivalent. A diamond (◊) appears next to windows that have been modified.